Principles of Concurrent and Distributed Programming

Emilio Tuosto

Academic Year 2025/2026

January 2026

On using the “right” primitives

Advanced primitives for concurrency

Join patterns are very high-level

Based on the join calculuc [FG96]

Integrated in some programming languages (Erlang, C$\#$, etc.)

We'll see a combination of join patterns and actors

- Novel specification of **fair join pattern matching** for actors
- Novel **stateful tree-based** matching algorithm with **proof of correctness**
- `JoinActors`: novel Scala 3 library for actors with fair join pattern matching

What are Join Patterns?

- Coordination mechanism for **concurrent** message passing programs
- Introduced in Join Calculus (Fournet et al., POPL 1996)

What are Join Patterns?

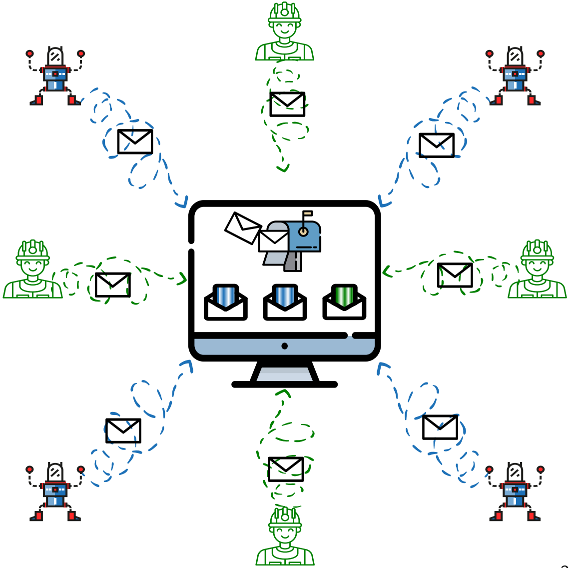
- Coordination mechanism for **concurrent** message passing programs
- Introduced in Join Calculus (Fournet et al., POPL 1996)
- Message passing programs may react to complex message sequences and conditions

What are Join Patterns?

- Coordination mechanism for **concurrent** message passing programs
- Introduced in Join Calculus (Fournet et al., POPL 1996)
- Message passing programs may react to complex message sequences and conditions
- Join patterns simplify specifying the **association of out-of-order messages**

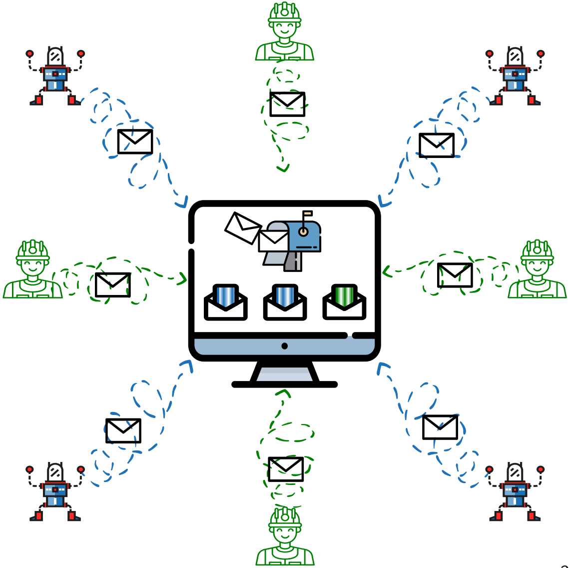
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers



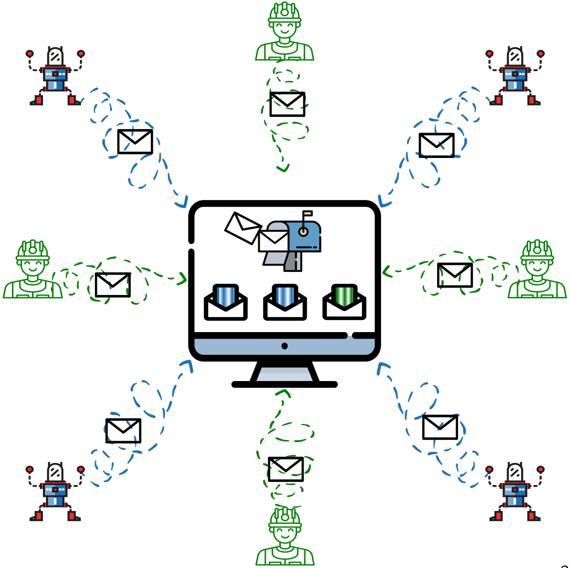
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**



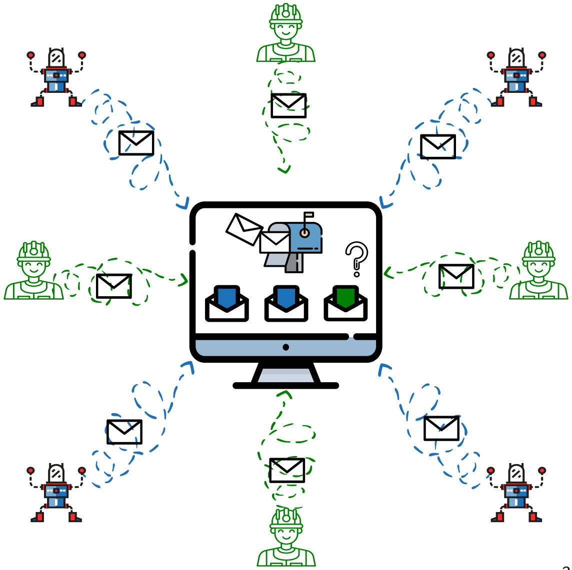
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**
- Monitor reacts to a combination of messages in the mailbox



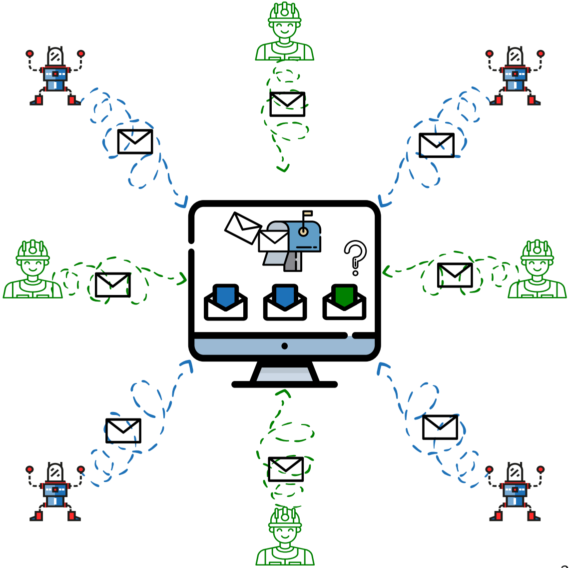
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**
- Monitor reacts to a combination of messages in the mailbox
- Traditionally, programmers write **custom code** for message association

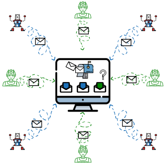


Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**
- Monitor reacts to a combination of messages in the mailbox
- Traditionally, programmers write **custom code** for message association (e.g., Akka/Pekko actors, Socket programming)



Factory Shop Monitor Using JoinActors

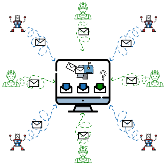


Using our `JoinActors` library we can **declaratively** specify **order-independent message associations**

```scala
def monitor() = Actor[...] {
  receive { (...) => {
    case (Fault(id1, _), Fix(id2, _)) if id1 == id2 => ...

    case (Fault(_, ts1), Fault(id2, ts2), Fix(id3, _))
                 if id2 == id3 && ts2 - ts1 > TEN_MIN => ...
  }}
}
```

- Uses Scala 3 macros

Join Patterns More Formally



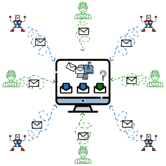
Let $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

Refer to the paper for more details

Join Patterns Matching



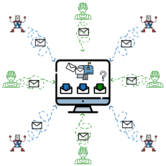
The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

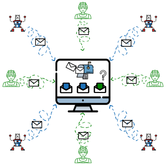
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} =$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

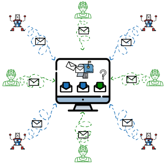
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

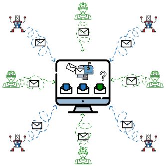
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

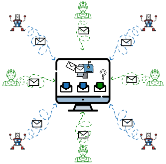
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

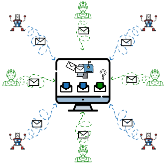
$$\Pi_1 = \mathtt{Fault}(id_1, \_) \wedge \mathtt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \mathtt{Fault}(\_, t_1) \wedge \mathtt{Fault}(id_2, t_2) \wedge \mathtt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \mathtt{Fault}_1\,(1, 10{:}35) \cdot \mathtt{Fault}_2\,(2, 10{:}40) \cdot \mathtt{Fault}_3\,(3, 10{:}55) \cdot \mathtt{Fix}_4\,(3, 11{:}00)$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

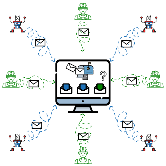
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1\,(1, 10{:}35) \cdot \texttt{Fault}_2\,(2, 10{:}40) \cdot \texttt{Fault}_3\,(3, 10{:}55) \cdot \texttt{Fix}_4\,(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$.

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

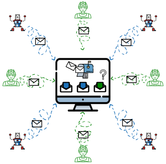
$$\Pi_2 = \texttt{Fault}(\_, t_1) \land \texttt{Fault}(id_2, t_2) \land \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

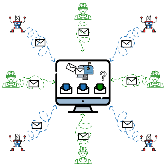
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**
  - $\Pi_1 : \langle\{\texttt{Fault}_3, \texttt{Fix}_4\}\rangle$

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

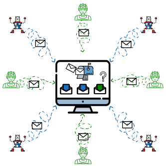
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**
  - $\Pi_1 : \langle\{\texttt{Fault}_3, \texttt{Fix}_4\}\rangle$
  - $\Pi_2 : \langle\{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\}\rangle$

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

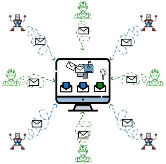
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \; \&\& \; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**
  - $\Pi_1 : \langle\{\texttt{Fault}_3, \texttt{Fix}_4\}\rangle$
  - $\Pi_2 : \langle\{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\}\rangle$
- In existing literature, the selection is either
  - **Non-deterministic** choice. This is usually undesirable
  - Pick **longest-matching sequence**

Our Proposal: "Fair Match"



Recall that we have the following $D = \Pi_1 + \Pi_2$ where:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

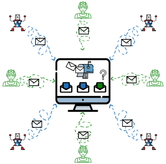
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

And the following final mailbox configuration:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- A "fair" match is the one that consumes the **oldest** messages in $\mathcal{M}$
- No message that can be matched is left in the mailbox **indefinitely**

Our Proposal: "Fair Match"



Recall that we have the following $D = \Pi_1 + \Pi_2$ where:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

And the following final mailbox configuration:

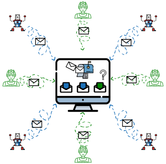
$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- A "fair" match is the one that consumes the **oldest** messages in $\mathcal{M}$
- No message that can be matched is left in the mailbox **indefinitely**
- Now we can pick the **fairest** match from $\mathcal{M}$:

$$\Pi_1 : \langle \{\texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

$$\Pi_2 : \langle \{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

Our Proposal: “Fair Match”



Recall that we have the following $D = \Pi_1 + \Pi_2$ where:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

And the following final mailbox configuration:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- A “fair” match is the one that consumes the **oldest** messages in $\mathcal{M}$
- No message that can be matched is left in the mailbox **indefinitely**
- Now we can pick the **fairest** match from $\mathcal{M}$:

$$\Pi_1 : \langle \boxed{\{\texttt{Fault}_3, \texttt{Fix}_4\}} \rangle$$

$$\Pi_2 : \langle \boxed{\{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

$$D : \langle \{\texttt{Fault}_3, \texttt{Fix}_4\}, \boxed{\{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}} \rangle$$

"Fair" Match Formalisation

We have formalised this notion of "fair" join pattern matching declaratively using inference rules:

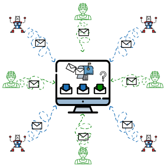
$$\frac{\forall i \in \{1, \ldots, n\} : \mu_i\sigma = m_i \qquad \gamma\sigma}{m_1 \cdot \ldots \cdot m_n \vDash_\sigma \mu_1 \wedge \ldots \wedge \mu_n \text{ if } \gamma} \; \textbf{Match Messages Against Pattern}$$

$$\frac{\mathcal{M}[\mathcal{I}] \vDash_\sigma \Pi \text{ for some } \sigma}{\mathcal{M} \vDash_\mathcal{I} \Pi} \; \textbf{Pick Messages From } \mathcal{M}$$

$$\frac{\mathcal{M} \vDash_\mathcal{I} \Pi \qquad \forall \mathcal{I}'. \, (\mathcal{M} \vDash_{\mathcal{I}'} \Pi \implies \mathcal{I} \leqslant_{\text{lex}} \mathcal{I}')}{\mathcal{M} \vDash \Pi \rightsquigarrow \mathcal{I}} \; \textbf{Select Fairest Match}$$

- Translate inference rules into a "fair" message matching **brute-force algorithm**
- Current implementations use matching without fairness e.g. (Haller et al. COORDINATION 2008, Plociniczak and Eisenbach COORDINATION 2010, Avila et al. 2020)
- Refer to the paper for more details

Brute-force Algorithm for "Fair" Message Matching



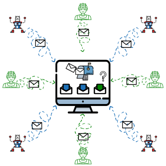
Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} =$$

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts

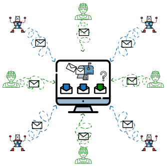
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot$$

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

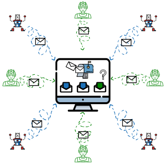
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_)\,\cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts

We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

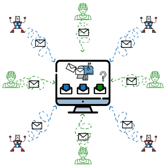
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1] : \langle \texttt{Fix}_1(3, \_) \rangle$ – Not enough messages ✕

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

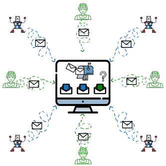
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1 (3, \_) \cdot \texttt{Fault}_2 (1, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1 (3, \_) \rangle$ – Not enough messages ×

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

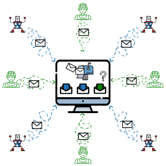
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_)\rangle$ – Not enough messages ×
$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_)\rangle$

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

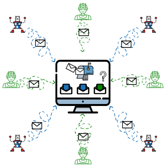
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot \texttt{Fault}_3\,(2, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_) \rangle$ – Not enough messages ×
$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot \texttt{Fault}_3\,(2, \_) \cdot$$

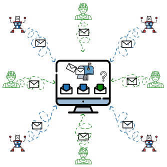
- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_) \rangle$ – Not enough messages ✕

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$:

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot \texttt{Fault}_3\,(2, \_) \cdot$$

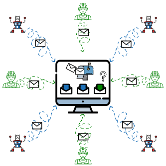
- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_) \rangle$ – Not enough messages ✕

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot \texttt{Fault}_3\,(2, \_) \cdot$$

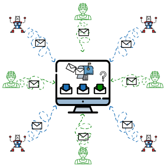
- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_)\rangle$ – Not enough messages ×

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_)\rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_)\rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_3\,(2, \_)\rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot \texttt{Fault}_3\,(2, \_) \cdot \texttt{Fault}_4\,(3, \_)$$

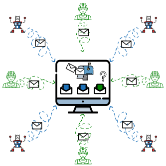
- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_) \rangle$ – Not enough messages ✕

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_3\,(2, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot \texttt{Fault}_3\,(2, \_) \cdot \texttt{Fault}_4\,(3, \_)$$

- Find a match for Π_1 from $\mathcal{M}$

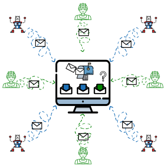
$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_) \rangle$ – Not enough messages ✕

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_3\,(2, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3 \cdot 4]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_3\,(2, \_) \rangle$,

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot \texttt{Fault}_3\,(2, \_) \cdot \texttt{Fault}_4\,(3, \_)$$

- Find a match for Π_1 from $\mathcal{M}$

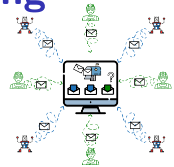
$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_) \rangle$ – Not enough messages ×

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_3\,(2, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3 \cdot 4]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_3\,(2, \_) \rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_4\,(3, \_) \rangle$

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

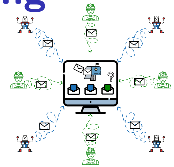
and the following mailbox:

$$\mathcal{M} =$$

$\emptyset$

Check if $id_1 = id_2$

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

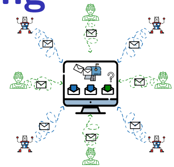
and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot$$

$\emptyset$

Check if $id_1 = id_2$

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

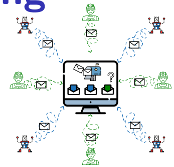
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot$$

$\emptyset$
└ $\{\texttt{Fault}_1\}$

Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

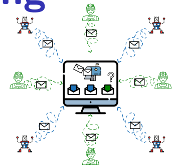
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot$$

$\emptyset$
└ $\{\texttt{Fault}_1\}$

Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

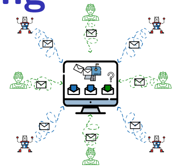
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot$$

$\emptyset$
- $\{\texttt{Fault}_1\}$
- $\{\texttt{Fault}_2\}$

Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

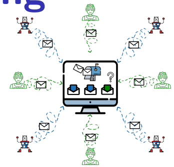
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot$$

$\emptyset$
- $\{\texttt{Fault}_1\}$
- $\{\texttt{Fault}_2\}$

Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

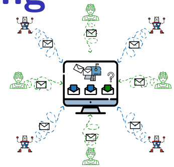
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
  - $\{\texttt{Fault}_2\}$
  - $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

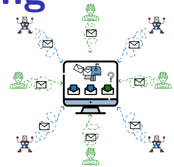
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

$\emptyset$
- $\{\texttt{Fault}_1\}$
- $\{\texttt{Fault}_2\}$
- $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

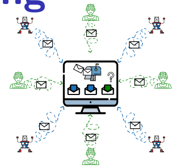
and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\{\texttt{Fault}_1, \texttt{Fix}_4\}$
  - $\{\texttt{Fault}_2\}$
  - $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

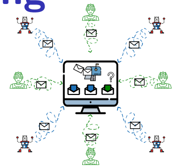
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\{\texttt{Fault}_1, \texttt{Fix}_4\}$
  - $\{\texttt{Fault}_2\}$
  - $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

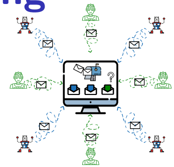
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

$\emptyset$
- $\{\texttt{Fault}_1\}$
  - $\{\texttt{Fault}_1, \texttt{Fix}_4\}$
- $\{\texttt{Fault}_2\}$
- $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ✗

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

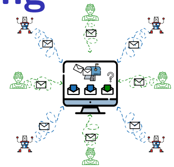
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

$\emptyset$
- $\{\texttt{Fault}_1\}$
  - $\boxed{\{\texttt{Fault}_1, \texttt{Fix}_4\} \times}$
- $\{\texttt{Fault}_2\}$
- $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

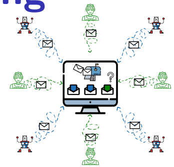
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

$\emptyset$
- $\{\texttt{Fault}_1\}$
  - $\boxed{\{\texttt{Fault}_1, \texttt{Fix}_4\} \times}$
- $\{\texttt{Fault}_2\}$
  - $\{\texttt{Fault}_2, \texttt{Fix}_4\}$
- $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ $\times$

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

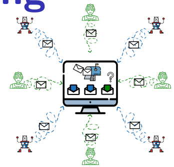
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\boxed{\{\texttt{Fault}_1, \texttt{Fix}_4\} \times}$
  - $\{\texttt{Fault}_2\}$
    - $\{\texttt{Fault}_2, \texttt{Fix}_4\}$
  - $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

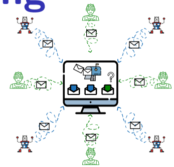
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

$\emptyset$
- $\{\texttt{Fault}_1\}$
  - $\boxed{\{\texttt{Fault}_1, \texttt{Fix}_4\} \times}$
- $\{\texttt{Fault}_2\}$
  - $\{\texttt{Fault}_2, \texttt{Fix}_4\}$
- $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

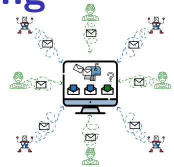
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

$\emptyset$
- $\{\texttt{Fault}_1\}$
  - $\boxed{\{\texttt{Fault}_1, \texttt{Fix}_4\} \times}$
- $\{\texttt{Fault}_2\}$
  - $\boxed{\{\texttt{Fault}_2, \texttt{Fix}_4\} \times}$
- $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \mathtt{Fault}(id_1, \_) \land \mathtt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

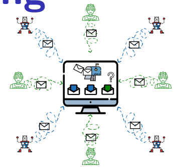
$$\mathcal{M} = \mathtt{Fault}_1\,(1, \_) \cdot \mathtt{Fault}_2\,(2, \_) \cdot \mathtt{Fault}_3\,(3, \_) \cdot \mathtt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\mathtt{Fault}_1\}$
    - $\{\mathtt{Fault}_1, \mathtt{Fix}_4\}$ ×
  - $\{\mathtt{Fault}_2\}$
    - $\{\mathtt{Fault}_2, \mathtt{Fix}_4\}$ ×
  - $\{\mathtt{Fault}_3\}$
    - $\{\mathtt{Fault}_3, \mathtt{Fix}_4\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

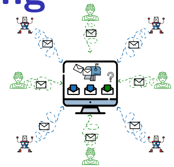
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\boxed{\{\texttt{Fault}_1, \texttt{Fix}_4\} \times}$
  - $\{\texttt{Fault}_2\}$
    - $\boxed{\{\texttt{Fault}_2, \texttt{Fix}_4\} \times}$
  - $\{\texttt{Fault}_3\}$
    - $\{\texttt{Fault}_3, \texttt{Fix}_4\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×
- **Attempt 3**: $3 = 3$

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

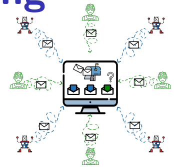
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\boxed{\{\texttt{Fault}_1, \texttt{Fix}_4\} \times}$
  - $\{\texttt{Fault}_2\}$
    - $\boxed{\{\texttt{Fault}_2, \texttt{Fix}_4\} \times}$
  - $\{\texttt{Fault}_3\}$
    - $\{\texttt{Fault}_3, \texttt{Fix}_4\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×
- **Attempt 3**: $3 = 3$ ✓

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

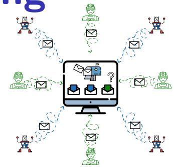
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\boxed{\{\texttt{Fault}_1, \texttt{Fix}_4\} \times}$
  - $\{\texttt{Fault}_2\}$
    - $\boxed{\{\texttt{Fault}_2, \texttt{Fix}_4\} \times}$
  - $\{\texttt{Fault}_3\}$
    - $\boxed{\{\texttt{Fault}_3, \texttt{Fix}_4\} \checkmark}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×
- **Attempt 3**: $3 = 3$ ✓

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

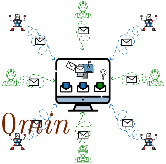
$\emptyset$
- $\{\texttt{Fault}_1\}$
  - $\{\texttt{Fault}_1, \texttt{Fix}_4\}$ ✗
- $\{\texttt{Fault}_2\}$
  - $\{\texttt{Fault}_2, \texttt{Fix}_4\}$ ✗
- $\{\texttt{Fault}_3\}$
  - $\{\texttt{Fault}_3, \texttt{Fix}_4\}$ ✓
- $\{\texttt{Fix}_4\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ✗
- **Attempt 2**: $2 \neq 3$ ✗
- **Attempt 3**: $3 = 3$ ✓

We don't record a partial match $\texttt{Fix}_4$ because we matched earlier

Tree Construction (continued)



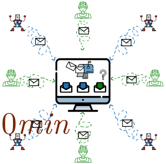
We now consider the second join pattern Π_2:

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1\,(1, 10{:}35) \cdot \texttt{Fault}_2\,(2, 10{:}40) \cdot \texttt{Fault}_3\,(3, 10{:}55) \cdot \texttt{Fix}_4\,(3, 11{:}00)$$

Tree Construction (continued)



We now consider the second join pattern Π_2:

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\{\texttt{Fault}_1, \texttt{Fault}_2\}$
      - $\{\texttt{Fault}_1, \texttt{Fault}_2, \texttt{Fix}_4\}$ ✗
    - $\{\texttt{Fault}_1, \texttt{Fault}_3\}$
      - $\{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}$ ✓
  - $\{\texttt{Fault}_2\}$
    - $\{\texttt{Fault}_2, \texttt{Fault}_3\}$
      - $\{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\}$
  - $\{\texttt{Fault}_3\}$
    - $\{\texttt{Fault}_3, \texttt{Fix}_4\}$
  - $\{\texttt{Fix}_4\}$

Check if $id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$:

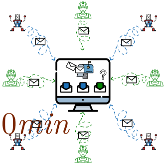
- **Attempt 1**:

  $1 = 3 \;\&\&\; 10:40 - 10:35 > 10min$ ✗

- **Attempt 2**:

  $3 = 3 \;\&\&\; 10:55 - 10:35 > 10min$ ✓

Tree Construction (continued)



We now consider the second join pattern Π_2:

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\{\texttt{Fault}_1, \texttt{Fault}_2\}$
      - $\{\texttt{Fault}_1, \texttt{Fault}_2, \texttt{Fix}_4\}$ ✗
    - $\{\texttt{Fault}_1, \texttt{Fault}_3\}$
      - $\{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}$ ✓
  - $\{\texttt{Fault}_2\}$
    - $\{\texttt{Fault}_2, \texttt{Fault}_3\}$
      - $\{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\}$
  - $\{\texttt{Fault}_3\}$
    - $\{\texttt{Fault}_3, \texttt{Fix}_4\}$
  - $\{\texttt{Fix}_4\}$

Check if $id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$:

- **Attempt 1**:

  $1 = 3 \;\&\&\; 10:40 - 10:35 > 10min$ ✗
- **Attempt 2**:

  $3 = 3 \;\&\&\; 10:55 - 10:35 > 10min$ ✓

We avoid computing (partial) matches

Performance Evaluation

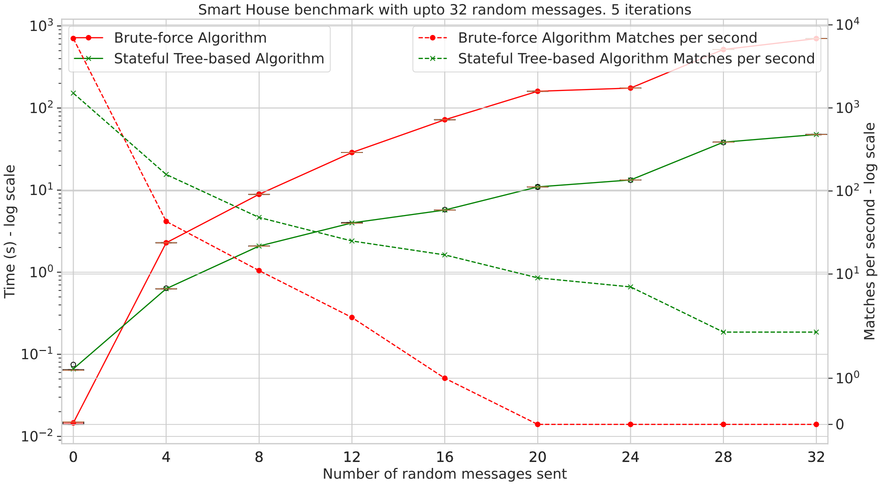


Figure: Smart House benchmark based on (Rodriguez-Avila et al. 2021)

Contributions & Future Work

Contributions:

- Novel specification of **fair and deterministic join pattern matching**
- Novel **stateful tree-based matching algorithm** to avoid redundant recomputations
- **Proof of correctness** of the stateful fair matching algorithm
- `JoinActors`: novel Scala 3 library with brute-force & stateful matching
- Established a **benchmark suite** to evaluate join pattern matching performance

Contributions & Future Work

Contributions:

- Novel specification of **fair and deterministic join pattern matching**
- Novel **stateful tree-based matching algorithm** to avoid redundant recomputations
- **Proof of correctness** of the stateful fair matching algorithm
- `JoinActors`: novel Scala 3 library with brute-force & stateful matching
- Established a **benchmark suite** to evaluate join pattern matching performance

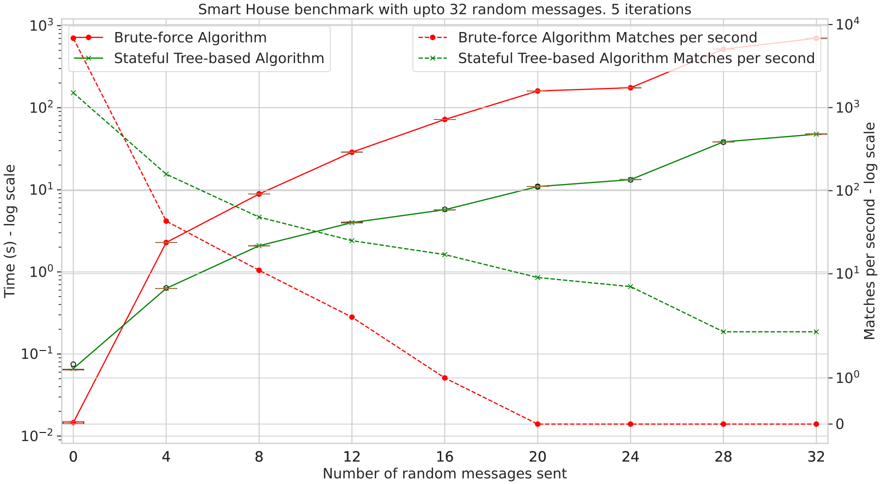
Future Work:

- Expand benchmark suite with more examples from the literature
- Refine and optimise the Scala 3 implementation of join patterns
- Alternative matching policies
- Verify join pattern unreachablity

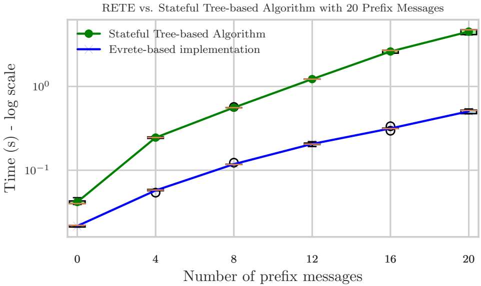
Smart House Example (Rodriguez-Avila et al. 2021) I

```
case (Motion(_, mStatus, mRoom, t0),
      AmbientLight(_, value, alRoom, t1),
      Light(_, lStatus, lRoom, t2)) if bathroomOccupied(...) => ...

case (Motion(_, mStatus0, mRoom0, t0),
      Contact(_, cStatus, cRoom, t1),
      Motion(_, mStatus1, mRoom1, t2)) if occupiedHome(...) => ...

case (Motion(_, mStatus0, mRoom0, t0),
      Contact(_, cStatus, cRoom, t1),
      Motion(_, mStatus1, mRoom1, t2)) if emptyHome(...) => ...
```

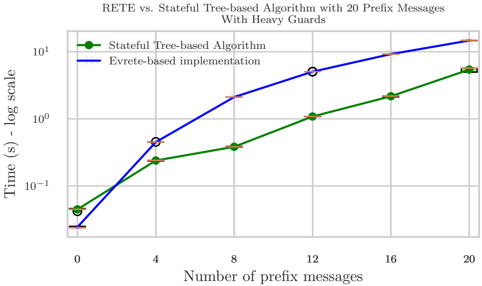
Smart House Example (Rodriguez-Avila et al. 2021) II



JoinActors vs. Evrete Benchmark



JoinActors vs. Evrete (lower is better)



JoinActors vs. Evrete (lower is better)

- Evrete is a mature and highly optimised RETE-based Java rule engine library
- JoinActors is our proof-of-concept Scala 3 actor library

Join Patterns Implementation in Scala 3

```scala
inline def receive[M, T](
    inline f: ActorRef[M] => PartialFunction[Any, Result[T]]
): MatchingAlgorithm => Matcher[M, Result[T]]
```

Macro Expansion & Code Transformation

The body of receive:

```
...
expr.asTerm match
    case Inlined(_, _, Block(_, Block(stmts, _))) =>
      stmts.head match
        case DefDef(_, List(TermParamClause(params)), _, Some(Block(_,
          ↪ Block(body, _)))) =>
          body.head match
            case DefDef(_, _, _, Some(Match(_, cases))) =>
              cases.flatMap { generateJoinPattern[M, T](_) }
...
```

A problem in concurrency [Tro94]

Problem Definition

Santa Claus sleeps in his shop up at the North Pole, and can only be wakened by either all nine reindeer being back from their year long vacation on the beaches of some tropical island in the South Pacific, or by some elves who are having some difficulties making the toys. One elf's problem is never serious enough to wake up Santa (otherwise, he may **never** get any sleep), so, the elves visit Santa in a group of three. When three elves are having their problems solved, any other elves wishing to visit Santa must wait for those elves to return. If Santa wakes up to find three elves waiting at his shop's door, along with the last reindeer having come back from the tropics, Santa has decided that the elves can wait until after Christmas, because it is more important to get his sleigh ready as soon as possible. (It is assumed that the reindeer don't want to leave the tropics, and therefore they stay there until the last possible moment. They might not even come back, but since Santa is footing the bill for their year in paradise ... This could also explain the quickness in their delivering of presents, since the reindeer can't wait to get back to where it is warm.) The penalty for the last reindeer to arrive is that it must get Santa while the others wait in a warming hut before being harnessed to the sleigh.

A Solution

The solution that has worked best over the years, and also appears to be the simplest, is written using C statements and pseudo-code. (Constants are also used in case the number of reindeer were to change, or if the group size of "solution-seeking" elves is modified.) Basically, the reindeer arrive, update the count of how many have arrived, and the last one wakes up Santa. An elf, upon discovering a problem, attempts to modify the count for the number of elves with a problem and either: waits outside Santa's shop if he/she is the first or second such elf; knocks on the door and wakes up Santa if that elf is the third one; or waits in the elves' shop until the elves currently with Santa start coming back. (The code for this solution can be found in the Appendix.)

```
receive
    {reindeer, Pid1} and {reindeer, Pid2} and {reindeer, Pid3}
      and {reindeer, Pid4} and {reindeer, Pid5} and {reindeer, Pid6}
      and {reindeer, Pid7} and {reindeer, Pid8} and {reindeer, Pid9} ->
        io:format("Ho, ho, ho! Let's deliver presents!~n"),
        [Pid1, Pid2, Pid3, Pid4, Pid5, Pid6, Pid7, Pid8, Pid9];
    {elf, Pid1} and {elf, Pid2} and {elf, Pid3} ->
        io:format("Ho, ho, ho! Let's discuss R&D possibilities!~n"),
        [Pid1, Pid2, Pid3]
end
```

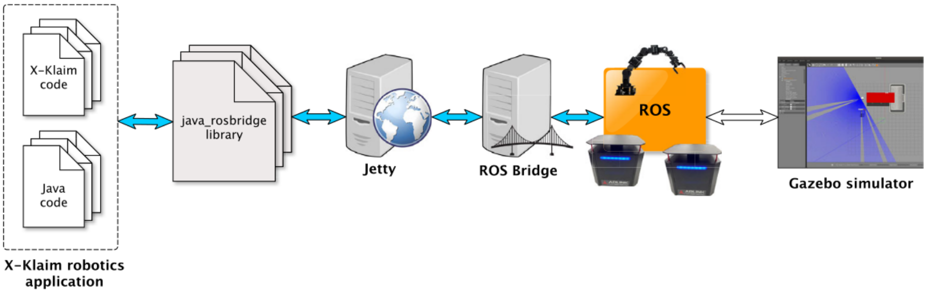
The solution with semaphores takes about 2 pages of C code [Tro94]!

Applying Concurrency with generative communication [CG89]

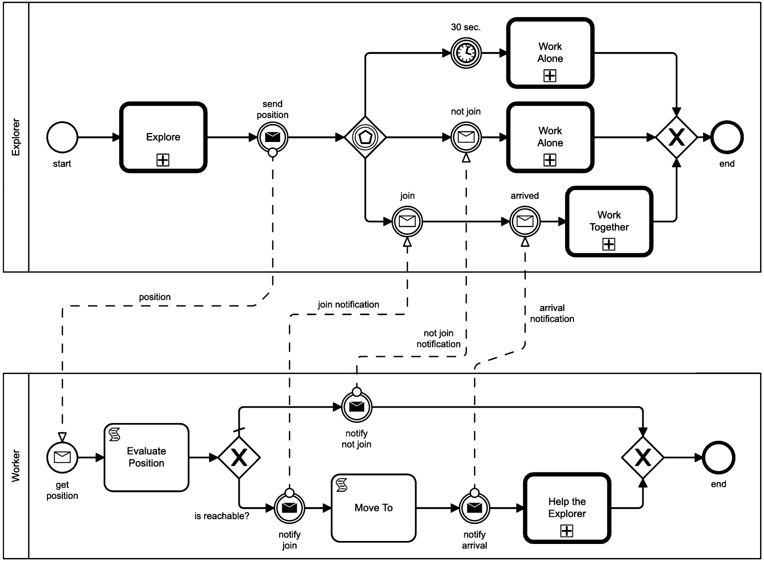
A model-driven approach for multi-robots missions [BTBS26]

Multi-robot application are complex: robots' interactions are "low-level"

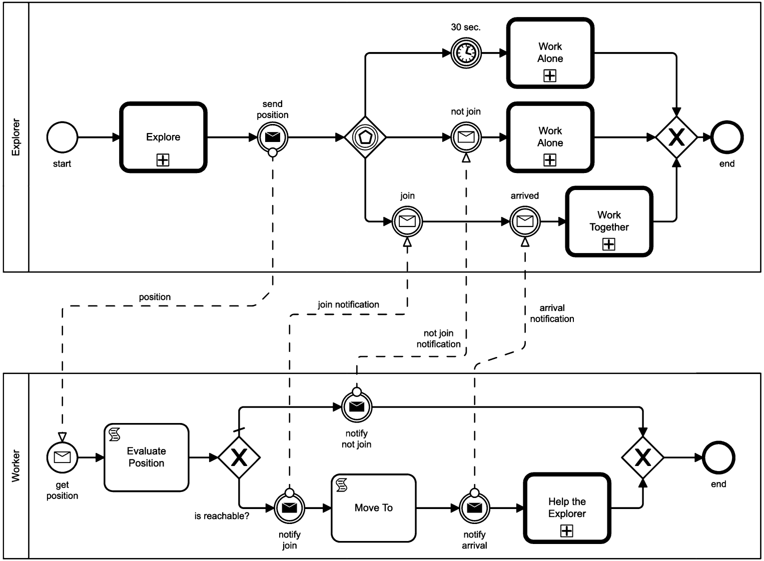
Model-driven development based on BPMN and X-KLAIM lowers barries



Business Process Modelling Notation

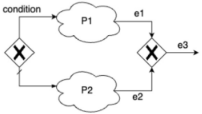


Business Process Modelling Notation



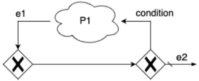
From BPMN to X-Klaim [BTBS26]

XOR



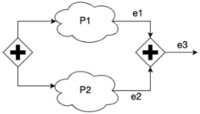

```
if(condition){
  translate(P1)
  in(e1)@self }
else{
  translate(P2)
  in(e2)@self }
out(e3)@self
```

Loop




```
while(condition){
  translate(P1)
  in(e1)@self }
out(e2)@self
```

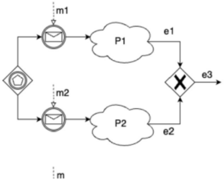
AND




```
eval(new ProcP1())@self
eval(new ProcP2())@self
in(e1)@self
in(e2)@self
out(e3)@self
```

```
// Processes to be
// added to the node
proc ProcP1(){
  translate(P1)
}

proc ProcP2(){
  translate(P2)
}
```

Event-Based (between messages)




```
var eventOccured = false
while(!eventOccured){
  if(in(m1,vars1)@self within pollTimeout){
    eventOccured = true
    translate(P1)
    in(e1)@self }
  else if(in(m2,vars2)@self within pollTimeout){
    eventOccured = true
    translate(P2)
    in(e2)@self } }
out(e3)@self
```

Klaim

Network-aware programming and generative communication:
X-KLAIM: eXtended Kernel Language for Agents Interaction and Mobility

Network

```
net MRS {
node Drone { eval(new DroneBehavior(Tractor)) @ self }
node Tractor { eval(new TractorBehavior()) @ self }
}
```

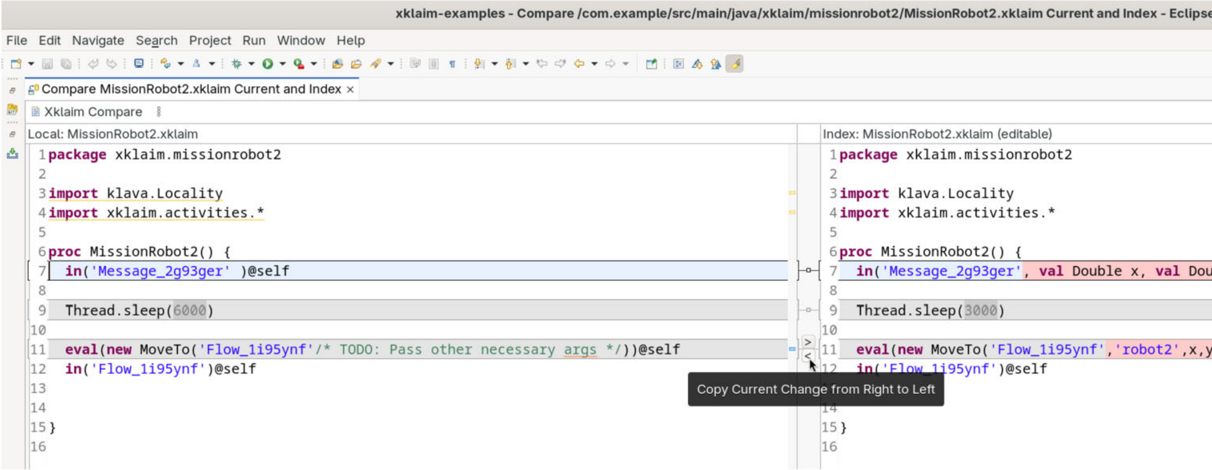
where

Some processes

```
proc DroneBehavior(Locality Tractor) {
eval(new WeedHandler(Tractor)) @ self
eval(new TakeOff("e1"))tractor @ self
in("e1") @ self
eval(new Explore("e2")) @ self
in("e2") @ self
eval(new Land("e3")) @ self
in("e3") @ self
}
```

```
proc TractorBehavior() {
in(WEED_POSITION, var Double x, var Double y) @ self
eval(new MoveTo("e4", x, y)) @ self
in("e4") @ self
eval(new CutGrass("e5")) @ self
in("e5") @ self
eval(new ReturnToBase("e6")) @ self
in("e6") @ self
```

Programming support



[BTBS26] Khalid Bourr, Francesco Tiezzi, Lorenzo Bettini, and Stefano Seriani. Translating bpmn models into x-klaim programs for developing multi-robot missions. *International Journal on Software Tools for Technology Transfer*, pages 1433–2787, January 2026.

[CG89] Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, April 1989.

[FG96] Cedric Fournet and George Gonthier. The reflexive CHAM and the join-calculus. In *Conference Record of POPL '96: The* 23$^{\mathrm{rd}}$ *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 372–385, St. Petersburg Beach, Florida, January 1996.

[Tro94] John A. Trono. A new exercise in concurrency. *SIGCSE Bull.*, 26(3):8–10, September 1994.